METHODS AND APPARATUS FOR LOADING A VERY LONG INSTRUCTION WORD MEMORY

Related Applications

The present invention claims the benefit of U.S. Provisional Application Serial No. 60/171,911 entitled "Methods and Apparatus for Loading a Very Long Instruction Word Memory" and filed December 23, 1999 which is incorporated by reference herein in its entirety.

Field of the Invention

The present invention relates generally to improvements in parallel processing, and more particularly to techniques for the loading of very long instruction word (VLIW) memory in an indirect VLIW processor which allows the load latency to be hidden by computation.

Background of the Invention

An indirect VLIW processor is organized with a VLIW memory (VIM) that is separate from its short instruction word (SIW) memory. The VIM is defined for an instruction set architecture by way of specific SIWs that control the loading and execution of the VLIWs stored in the VIM. For example, a load VLIW (LV) instruction is defined which acts as a setup control delimiter instruction for the processor logic. The LV instruction specifies the VIM address where a VLIW is to be stored and the number of SIWs which follow the LV instruction that are to be stored at the specified VIM address in VLIW fashion. Another special SIW is the execute VLIW (XV) instruction. The XV instruction causes a VLIW to be read out of VIM at the XV specified address.

The ManArray processor defines two preferred architectures for indirect VLIW memories. One approach treats the VIM as one composite block of memory using one common

address interface to access any VLIW stored in the VIM. The second approach treats the VIM as made up of multiple smaller memories each individually associated with the functional units and each individually addressable for loading and reading during XV execution. It will be recognized that improved techniques loading of VLIW memory will be highly desirable.

Summary of the Present Invention

The present invention covers techniques to independently load the VIMs concurrent with SIW or iVLIW execution on the SP or on the PEs thereby allowing the load latency to be hidden by the computation. The ManArray processor which is a scalable indirect VLIW array processor is the presently preferred processor for implementing these concepts. The VIM memories, contained in each processing element (PE), are accessible by the same type of LV and XV SIWs as in a single processor instantiation of the indirect VLIW architecture. In the ManArray architecture the control processor, also called the sequence processor (SP), fetches the instructions from the SIW memory and dispatches them to itself and the PEs. By using the LV instruction, VLIWs can be loaded into VIMs in the SP and the PEs. Since the LV instruction is supplied by the SP through the instruction stream, when VLIWs are being loaded into any VIM no other processing takes place. In addition, as defined in the ManArray architecture, when the SP is processing SIWs, such as control and other sequential code, the PE array is not executing any instructions. With the techniques presented herein, the latency to load the VIM can be hidden by the computation.

A more complete understanding of the present invention, as well as other features and advantages of the invention will be apparent from the following Detailed Description and the accompanying drawings.

Brief Description of the Drawings

Fig. 1 illustrates aspects of ManArray indirect VLIW instruction memory in accordance with the present invention;

Fig. 2 illustrates the basic iVLIW Data Path;

Fig. 3 illustrates a five slot iVLIW with an expanded view of the ALU slot;

Fig. 4A illustrates an LV1 Load/Modify VLIW Instruction;

Fig. 4B illustrates an XV1 Execute VLIW Instruction;

Fig. 4C illustrates an LV2 Load/Modify Instruction;

Fig. 4D illustrates an XV2 Execute VLIW Instruction;

Fig. 5 illustrates aspects of an iVLIW LV1/XV1 pipeline apparatus;

Fig. 6 illustrates aspects of an iVLIW LV2/XV2 pipeline apparatus;

Fig. 7 illustrates a Type-1 VIM DMA apparatus;

Fig. 8 illustrates a Type-2 VIM DMA apparatus; and

Fig. 9 illustrates a Type-2 VIM DMA - ALU portion apparatus.

Detailed Description

The present invention may be applicable to a variety of processing and array designs; however, an exemplary and presently preferred architecture for use in conjunction with the present invention is the ManArray™ architecture. Further details of a presently preferred ManArray core, architecture, and instructions for use in conjunction with the present invention are found in U.S. Patent Application Serial No. 08/885,310 filed June 30, 1997, now U.S. Patent No. 6,023,753, U.S. Patent Application Serial No. 08/949,122 filed October 10, 1997, U.S. Patent Application Serial No. 09/169,255 filed October 9, 1998, U.S. Patent Application Serial No. 09/169,256 filed October 9, 1998, U.S. Patent Application Serial No. 09/169,072 filed

October 9, 1998, U.S. Patent Application Serial No. 09/187,539 filed November 6, 1998, U.S. Patent Application Serial No. 09/205,558 filed December 4, 1998, U.S. Patent Application Serial No. 09/215,081 filed December 18, 1998, U.S. Patent Application Serial No. 09/228,374 filed January 12, 1999 and entitled "Methods and Apparatus to Dynamically Reconfigure the Instruction Pipeline of an Indirect Very Long Instruction Word Scalable Processor", U.S. Patent Application Serial No. 09/238,446 filed January 28, 1999, U.S. Patent Application Serial No. 09/267,570 filed March 12, 1999, U.S. Patent Application Serial No. 09/337,839 filed June 22, 1999, U.S. Patent Application Serial No. 09/350,191 filed July 9, 1999, U.S. Patent Application Serial No. 09/422,015 filed October 21, 1999 entitled "Methods and Apparatus for Abbreviated Instruction and Configurable Processor Architecture", U.S. Patent Application Serial No. 09/432,705 filed November 2, 1999 entitled "Methods and Apparatus for Improved Motion Estimation for Video Encoding", U.S. Patent Application Serial No. 09/471,217 filed December 23, 1999 entitled "Methods and Apparatus for Providing Data Transfer Control", U.S. Patent Application Serial No. 09/472,372 filed December 23, 1999 entitled "Methods and Apparatus for Providing Direct Memory Access Control", U.S. Patent Application Serial No. 09/596,103 entitled "Methods and Apparatus for Data Dependent Address Operations and Efficient Variable Length Code Decoding in a VLIW Processor" filed June 16, 2000, U.S. Patent Application Serial No. 09/598,567 entitled "Methods and Apparatus for Improved Efficiency in Pipeline Simulation and Emulation" filed June 21, 2000, U.S. Patent Application Serial No. 09/598,564 entitled "Methods and Apparatus for Initiating and Resynchronizing Multi-Cycle SIMD Instructions" filed June 21, 2000, U.S. Patent Application Serial No. 09/598,566 entitled "Methods and Apparatus for Generalized Event Detection and Action Specification in a Processor" filed June 21, 2000, and U.S. Patent Application Serial No. 09/598,084 entitled "Methods and Apparatus

for Establishing Port Priority Functions in a VLIW Processor" filed June 21, 2000, U.S. Patent Application Serial No. 09/599,980 entitled "Methods and Apparatus for Parallel Processing Utilizing a Manifold Array (ManArray) Architecture and Instruction Syntax" filed June 22, 2000, as well as, Provisional Application Serial No. 60/113,637 entitled "Methods and Apparatus for Providing Direct Memory Access (DMA) Engine" filed December 23, 1998, Provisional Application Serial No. 60/113,555 entitled "Methods and Apparatus Providing Transfer Control" filed December 23, 1998, Provisional Application Serial No. 60/139,946 entitled "Methods and Apparatus for Data Dependent Address Operations and Efficient Variable Length Code Decoding in a VLIW Processor" filed June 18, 1999, Provisional Application Serial No. 60/140,245 entitled "Methods and Apparatus for Generalized Event Detection and Action Specification in a Processor" filed June 21, 1999, Provisional Application Serial No. 60/140,163 entitled "Methods and Apparatus for Improved Efficiency in Pipeline Simulation and Emulation" filed June 21, 1999, Provisional Application Serial No. 60/140,162 entitled "Methods and Apparatus for Initiating and Re-Synchronizing Multi-Cycle SIMD Instructions" filed June 21, 1999, Provisional Application Serial No. 60/140,244 entitled "Methods and Apparatus for Providing One-By-One Manifold Array (1x1 ManArray) Program Context Control" filed June 21, 1999, Provisional Application Serial No. 60/140,325 entitled "Methods and Apparatus for Establishing Port Priority Function in a VLIW Processor" filed June 21, 1999, Provisional Application Serial No. 60/140,425 entitled "Methods and Apparatus for Parallel Processing Utilizing a Manifold Array (ManArray) Architecture and Instruction Syntax" filed June 22, 1999, Provisional Application Serial No. 60/165,337 entitled "Efficient Cosine Transform Implementations on the ManArray Architecture" filed November 12, 1999, and Provisional Application Serial No. 60/171,911 entitled "Methods and Apparatus for DMA Loading of Very

Long Instruction Word Memory" filed December 23, 1999, Provisional Application Serial No. 60/184,668 entitled "Methods and Apparatus for Providing Bit-Reversal and Multicast Functions Utilizing DMA Controller" filed February 24, 2000, Provisional Application Serial No. 60/184,529 entitled "Methods and Apparatus for Scalable Array Processor Interrupt Detection and Response" filed February 24, 2000, Provisional Application Serial No. 60/184,560 entitled "Methods and Apparatus for Flexible Strength Coprocessing Interface" filed February 24, 2000, Provisional Application Serial No. 60/203,629 entitled "Methods and Apparatus for Power Control in a Scalable Array of Processor Elements" filed May 12, 2000, and Provisional Application Serial No. 60/241,940 entitled "Methods and Apparatus for Efficient Vocoder Implementations" filed October 20, 2000, all of which are assigned to the assignee of the present invention and incorporated by reference herein in their entirety.

In a preferred embodiment of the present invention, a ManArray 2x2 iVLIW single instruction multiple data stream (SIMD) processor 100 illustrated in Fig. 1 is employed. Processor 100 includes a controller sequence processor (SP) combined with a processing element-0 (PE0) SP/PE0 101 as described in further detail in U.S. Patent Application Serial No. 09/169,072 entitled "Methods and Apparatus for Dynamic Merging an Array Controller with an Array Processing Element". Three additional PEs 151, 153, and 155 are used to describe the DMA and other techniques of loading of VIM memories. The SP/PE0 101 contains a fetch controller 103 to allow the fetching of SIWs from a 32-bit instruction memory 105. The fetch controller 103 provides the typical functions needed in a programmable processor such as a program counter (PC), a branch capability, event point loop operations (for further details of such operations see U.S. Provisional Application Serial No. 60/140,245 entitled "Methods and Apparatus for Generalized Event Detection and Action Specification in a Processor" filed June

21, 1999), and support for interrupts. It also provides the instruction memory control which could include an instruction cache if needed by an application. In addition, the SIW I-Fetch controller 103 dispatches 32-bit SIWs to the other PEs in the system by means of a 32-bit instruction bus 102.

In this exemplary system, common elements are used throughout to simplify the explanation, although actual implementations are not limited to this restriction. For example, the execution units 131 in the combined SP/PE0 101 can be separated into a set of execution units optimized for the control function, for example, fixed point execution units, and the PE0 as well as the other PEs can be optimized for a floating point application. For the purposes of the present description, it is assumed that the execution units 131 are of the same type in the SP/PE0 101 and the PEs 151, 153 and 155. In a similar manner, SP/PE0 and the other PEs use a five instruction slot iVLIW architecture which contains a VLIW memory (VIM) 109 and an instruction decode and VIM controller function unit 107 which receives instructions as dispatched from the SP/PE0's I-Fetch unit 103 and generates the VIM addresses-and-control signals 108 required to access the iVLIWs stored in the VIM 109. The iVLIWs are identified by the letters SLAMD that reference the Store(S), Load (L), ALU (A), MAU (M), and DSU (D) execution units. The loading of the iVLIWs is described in further detail in U.S. Patent Application Serial No. 09/187,539 entitled "Methods and Apparatus for Efficient Synchronous MIMD Operations with iVLIW PE-to-PE Communication". Also contained in the SP/PE0 and the other PEs is a common PE configurable register file (CRF) 127 which is described in further detail in U.S. Patent Application Serial No. 09/169,255 entitled "Method and Apparatus for Dynamic Instruction Controlled Reconfiguration Register File with Extended Precision".

Due to the combined nature of the SP/PE0 101, the data memory interface controller 125 must handle the data processing needs of both the SP controller, with SP data in memory 121, and PE0, with PE0 data in memory 123. The SP/PE0 controller 125 also is the source of the data that is sent over the 32-bit or 64-bit broadcast data bus 126. The other PEs, 151, 153, and 155 contain common physical data memory units 123', 123'', and 123''' though the data stored in them is generally different as required by the local processing done on each PE. The interface to these PE data memories is also a common design in PEs 1, 2, and 3 and indicated by PE local memory and data bus interface logic 157, 157' and 157''. Interconnecting the PEs for data transfer communications is a cluster switch 171 described in further detail in U.S. Patent Application Serial Nos. 08/885,310 entitled "Manifold Array Processor", 08/949,122 entitled "Methods and Apparatus for Manifold Array Processing", and 09/169,256 entitled "Methods and Apparatus for ManArray PE-to-PE Switch Control". The interface to a host processor, other peripheral devices, external memory or the like can be done in many ways. The primary mechanism shown for completeness is contained in a DMA control unit 181 that provides a scalable ManArray data bus 183 that connects to devices and interface units external to the ManArray core. The DMA control unit 181 provides the data flow and bus arbitration mechanisms needed for these external devices to interface to the ManArray core memories including the VIM via the multiplexed bus interface symbolically represented by line 185 and internal DMA interfaces 173, 175, 177, and 179. All of the above noted patents are assigned to the assignee of the present invention and incorporated herein by reference in their entirety.

Indirect Very Long Instruction Word (iVLIW) Architecture

Each SP/PE0 101 and each PE 151, 153, 155 in the ManArray architecture shown in Fig. 1 contains a quantity of iVLIW memory (VIM) 109. Each VIM 109 contains space to hold

multiple VLIW Instructions, and each addressable VIM location is capable of storing up to eight short instruction words (SIWs). Current implementations allow each iVLIW instruction to contain up to five simplex instructions: one in each of the store unit (SU), load unit (LU), arithmetic logic unit (ALU), multiply-accumulate unit (MAU), and data-select unit (DSU). For example, an iVLIW instruction at some VIM address "i" contains the five instructions SLAMD as indicated in the VIM 109 of Fig. 1.

Fig. 2 shows the basic iVLIW instruction flow execution path 200 in which a fetched instruction is stored in an instruction register 20 which is connected to a VIM load and store control functional unit 22. The VIM load and store control functional unit provides the interface signals to a VIM 24. The output of the VIM 24 is pipelined to an iVLIW register 26.

Fig. 3 illustrates a five slot iVLIW VIM 300 with N entries. The expanded ALU slot view indicates a 32-bit storage space, though depending upon the implementation the specific number of bits required to represent an instruction can vary.

iVLIW instructions can be loaded into an array of PE VIMs collectively, or, by using special instructions to mask a PE or PEs, each PE VIM can be loaded individually. The iVLIW instructions in VIM are accessed for execution through the execute VLIW (XV) instruction, which when executed as a single instruction causes the simultaneous execution of the simplex instructions located at a specified VIM memory address.

Two basic instruction types are used to load/modify iVLIW memories, and to execute iVLIW instructions. They are:

1. Load/Modify VLIW Memory Address (LV1) instruction 400 shown in Fig. 4A, and
2. Execute VLIW (XV1) instruction 425 shown in Fig. 4B.

As shown in Fig. 4A, the LV1 instruction 400 is for 32-bit encodings as shown in illustrative encoding block 410. A presently preferred syntax/operation for this instruction is shown in syntax/operation block 420. The LV1 instruction 400 is used to load and/or disable individual instruction slots of the specified SP or PE VLIW Memory (VIM). The VIM address is computed as the sum of the base VIM address register Vb (V0 or V1) plus an unsigned 8-bit offset VIMOFFS shown in bits 0-7, the block of bits 411 of encoding block 410 of Fig. 4A. The VIM address must be in the valid range for the hardware configuration otherwise the operation of this instruction is undefined.

Any combination of individual instruction slots may be disabled via the disable slot parameter 'd={SLAMD}, where S=store unit (SU), L=load unit (LU), A=arithmetic logic unit (ALU), M=multiply-accumulate unit (MAU), and D=data select unit (DSU). A blank 'd=' parameter does not disable any slots. An instruction loaded into a slot marked by the disable slot parameter remains disabled when loaded.

The number of instructions to be loaded are specified utilizing an *InstrCnt* parameter. For the present implementation, valid values are 0-5. As specified by this parameter, the number instructions next following the LV1 are loaded into the specified VIM address. Further operational description of the load VLIW instruction can be found in U.S. Patent Application Serial No. 09/187,539.

The XV1 instruction 425 shown in Fig. 4B is also for 32-bit encoding as shown in illustrative encoding block 430. A presently preferred syntax/operation for this instruction is shown in syntax/operation block 435. The XV1 instruction 425 is one of a control group of instructions defined by group field bits 30 and 31 of encoding block 410, and is used to execute individual instruction slots of the specified SP's or PE's, as specified by bit 29, VLIW memory

(VIM). The VIM address is computed as the sum of a base VIM address register *Vb* (V0 or V1) plus an unsigned 8-bit offset *VIMOFFS* shown in bits 0-7, the block of bits 431 in encoding block 430 of Fig. 4B. The VIM address must be in the valid range for the hardware configuration otherwise the operation of this instruction is undefined. Further operational description of the execute VLIW instruction is also found in the above mentioned application.

LV1 and XV1 operations in accordance with the present invention are discussed with reference to Fig. 5 where aspects of the iVLIW load and fetch pipeline are described in connection with an iVLIW system 500. Among its other aspects, Fig. 5 shows a selection mechanism for allowing selection of instructions out of VIM memory. A fetched instruction is loaded into instruction register 1 (IR1) 510. Register 510 corresponds generally with instruction register 20 of Fig. 2. The output of IR1 is predecoded by predecoder 512 early in the pipeline cycle prior to loading values into instruction register 2 (IR2) 514. When the instruction in IR1 is a load iVLIW (LV1) with a non-zero instruction count, the predecoder 512 generates the LVc1 control signals 515, which are used to set up the LV1 operation cycle. The VIM address 511 is calculated by use of the specified Vb register 502 added by adder 504 with the offset value included in the LV1 instruction via path 503. The resulting VIM address 511 is stored in register 506 and passed through multiplexer 508 to address the VIM 516. VIM 516 corresponds generally to VIM 109 of Fig. 1.

Register 506 is required to hold the VIM address 507 during the LV1 operations. The VIM address 511 and LV1 control state allows the loading of the instructions received after the LV1 instruction into the VIM 516. At the end of the cycle in which the LV1 was received, the disable bits 10-17 shown in Fig 4A are loaded into a d-bits register 518 for use when loading instructions into the VIM 516. Upon receipt of the next instruction in IR1 510, which is to be

loaded into VIM 516, the appropriate control signal is generated depending upon the instruction type, Storec1 519, Loadc1 521, ALUc1 523, MAUc1 525, or DSUc1 527. The operation of predecoder 512 is based upon a simple decoding of the group bits (bits 30 and 31 of Figs 4A-D) which define the instruction type shown in Figs. 4A-D, a Load/Store bit, and the unit field (bits 27 and 28) of an ALU, MAU, or DSU type instruction. By using this predecode step, the instruction in IR1 510 can be loaded into VIM 516 in the proper functional unit position.

For example, when an ADD instruction included in the LV1 list of instructions is received into IR1 510, it can be determined by the predecoder 512 that the ADD instruction is to be loaded into the ALU Instruction slot 520 in VIM 516. In addition, the appropriate d-bit 531 for that functional slot position is loaded into bit-31 of that slot. The loaded d-bit occupies one of the group code bit positions from the original instruction.

Upon receipt of an XV1 instruction in IR1 510, the VIM address 511 is calculated by use of the specified Vb register 502 added by adder 504 with the offset value included in the XV1 instruction via path 503. The resulting VIM Address 507 is passed through multiplexer 508 to address the VIM. The iVLIW at the specified address is read out of the VIM 516 and passes through the multiplexers 530, 532, 534, 536, and 538, to the IR2 registers 514. As an alternative to minimize the read VIM access timing critical path, the output of the VIM can be latched into a register whose output is passed through a multiplexer prior to the decode state logic.

For XV1 execution, an IR2MUX1 control signal 533 in conjunction with the predecode XVc1 control signal 517 cause all the IR2 multiplexers, 530, 532, 534, 536, and 538, to select the VIM output paths, 541, 543, 545, 547, and 549. At this point, the five individual decode and execution stages of the pipeline, 540, 542, 544, 546, and 548, are completed in synchrony providing iVLIW parallel execution performance. To allow a single 32-bit instruction to

execute by itself in the PE or SP, the bypass VIM path 535 is shown. For example, when a simplex ADD instruction is received into IR1 510 for parallel array execution, the predecoder 512 generates the IR2MUX1 533 control signal, which in conjunction with the instruction type predecode signal 523 in the case of an ADD, and lack of an XV 517 or LV 515 active control signal, causes the ALU multiplexer 534 to select the bypass path 535.

An alternative VIM configuration exists for within slot compression mechanism. This mechanism is described in further detail in U.S. Patent Application Serial No. 09/205,558. In this approach, the VIM is divided up into separate VIM sections each associated with corresponding functional decode-and-execute units. Each of the VIMs' address maps are divided into multiple 4-bit addressable sections as governed by an offset field included in a new version of the execute iVLIW instruction, XV2, with a separate offset that can be specified for each VIM slot section. This VIM configuration and XV2 addressing option provides the ability to independently select instructions within each VIM slot 4-bit address range. By doing this, duplicate SIWs within the 16 addressable iVLIW range can be eliminated providing greater packing of SIWs within the composite VIM.

The XV2 instruction is similar to the XV1 instruction in that it is used to modify, enable/disable sub-iVLIW instructions, and indirectly execute iVLIW instructions in the SP and PEs but does so in a different way than the XV1 instruction. For the XV2 instruction, it is still assumed that the iVLIWs have been loaded into this new partitioned VIM by use of a new version of the load VLIW instruction, LV2 455 shown in Fig. 4C. The LV2 encoding 450 consists of a CtrlOp field, bits 25-28, that represent the LV2 instruction opcode. A load instruction bit-23 specifies if at least one instruction is to be loaded or if the disable d-bit for the specified address is to be loaded. Bit-22 is the disable d-bit that is loaded. Bits 18-21 specify

that up to 16 instructions are to be loaded in the specified functional unit's VIM, as specified by bits 15-17, beginning at the address specified by one of the Vb registered addresses, as selected by bit-9, plus the VIMOFFS offset address, bits 0-7. Presently preferred syntax/operation details are shown in syntax/operation table 460.

An XV2 instruction 475 is shown in Fig. 4D. The encoding format 470 includes new bit fields as follows. UAF field bits 23 and 24 are not optional on XV2 instruction 475 and must be specified with each XV2 use. The VIM base register selection Vb is established by bit 20, and the five offset fields are store VIM offset (SOFS) bits 16-19, load VIM offset (LOFS) bits 12-15, ALU VIM offset (AOFS) bits 8-11, MAU VIM offset (MOFS) bits 4-7, and DSU VIM offset (DOFS) bits 0-3. The presently preferred syntax/operation is shown in syntax/operation table 480.

Referring to Fig. 6, VIM 616 consists of multiple independent memory units each associated with their functional decode and execute units. Independent addressing logic is provided for each slot VIM. As illustrated in Fig. 6, each VIM entry preferably consists of five SIW slots (one per execution unit) and associated with each SIW slot are additional state bits, of which 5 are shown (one d-bit per slot). Included among the five execution units are a store unit 640 associated with store instruction VIM 620, load unit 642 associated with load instruction VIM 622, an arithmetic-logic unit (ALU) 644 associated with ALU instruction VIM 624, a multiply accumulate unit (MAU) 646 associated with MAU instruction VIM 626, and a data select unit (DSU) 648 associated with DSU instruction VIM 628.

The Fig. 6 VIM address adder functional blocks, as exemplified by ALU VIM address adder 604, are different than the adder functional block 504 as shown in Fig. 5 in order to support the VIM address increment capability required by the load VLIW-2 (LV2) instruction

455 of Fig. 4C as described in the syntax/operation block 460. This capability allows the instructions following the LV2 instruction to be loaded at:

(V[01]+VIMOFFS)[UnitVIM] ← 1st Instruction following LV2

(V[01]+VIMOFFS+1)[UnitVIM] ← 2nd Instruction following LV2

:

(V[01]+VIMOFFS+InstrCnt)[UnitVIM] ← (InstrCnt)th Instruction following LV2

The instruction count parameter InstrCnt is a binary coded number, 0 thru F, that represents from 1 to 16 instructions that can be loaded into up to 16 consecutive UnitVIM locations.

The five state d-bits 621, 623, 625, 627, and 629 are LV-loaded disable bits for the instruction slots that indicate either: the SIW slot is available-for-execution or it is not-available-for-execution. A binary value suffices to distinguish between the two states. An instruction slot with its d-bit set to the not-available-for-execution state is interpreted as an NOP (no-operation) instruction by the execution unit. In addition, the appropriate d-bit for that functional slot position is loaded into bit-31 of that slot.

ManArray DMA Background

The ManArray PEs 101, 151, 153, 155 of Fig. 1 each have their own local memories 123, 123', 123", and 123"', respectively. These PE local memories are accessible from a program through use of the PE load and store instructions. PE local memories may also be accessed by the DMA controller for data transfers to and from other system memories. The DMA controller 181 accesses a PE local memory bank only when the PE does not require access to the particular port requested by the DMA. This access approach is termed "cycle borrowing" since the processor is never stalled for DMA access. PE0 which shares its load and store units with the SP behaves like any other PE with respect to its local memory accesses.

Type-1 VIM DMA

Fig. 7 depicts a type-1 VIM DMA apparatus 700 with type-1 VIM 708 similar to the type-1 VIM apparatus 500 and VIM 516 shown in Fig. 5, but with the addition of DMA control apparatus as shown in Fig. 7. The type-1 VIM DMA apparatus represents an extension to the basic DMA memory interface provided in each PE. The "cycle borrowing" technique is also used in VIM DMA operations. The preferred VIM 708 is a two port memory allowing simultaneous read and write accesses. It is anticipated that in applications where performance is not critical or where simultaneous read and write accesses are deemed not necessary that a single shared read/write port VIM can be used. In the two port design, each port requires its own address and write or read control signals. A write address 731 is sourced from multiplexer 736 since the VIM can be loaded from either instruction path 727 with LV1 instructions or from DMA operations on path 729. Read address 725 is only accessed from the instruction side under control of XV1 instructions. The DMA operation consists of a "data" packet made up of SIWs that begin with an LV1 type instruction, followed by the SIWs constituting the VLIW to be loaded at the address specified in the LV1 instruction. Multiple VLIW packets can be loaded into VIM during a single VIM DMA request with each packet optionally varying in size depending upon the number of SIWs contained in each VLIW.

Operation is as follows. The "data" packet, when the DMA operation begins, is transferred beginning with the LV1 "data" item over the 32-bit DMA bus 715 and stored into DMA Register 1 (DR1) 710. The output 717 of DR1 is provided to the VIM controller 722 which provides "cycle borrow" control to the DMA interface. The received LV1 instruction causes the correct Vx base address register to be selected via path 726 and adder 724 adds the base Vx value to the offset found in the received LV1 OFFSET field 719 which is labeled

Doffset. The resultant sum is stored in register 730 for later use. A programmer can use the two Vx base address registers V0 and V1 to aid this process where one, say register V0, can be set up for the instruction iVLIW usage and the other, register V1, for DMA base address use. The DMA packet will continue sending the next LV1 SIWs one at a time through DR1 710 to be loaded into the correct slot position DMA register for the SU 712, LU 714, ALU 716, MAU 718, and DSU 720. At this point, the full VLIW "data" packet has been loaded into the PEs and the VIM controller 722 can allow the VLIW to be loaded at the LV1 specified address as long as the instruction side does not take priority. The VIM controller 722 selects the multiplexer 736 to the DMA VIM address 729 and selects the multiplexers 750-758 to the VMA registers and in a single cycle the full VLIW is then loaded into the VIM. The DMA interface is then allowed to proceed with the next VLIW "data" packet if there is one.

Type-2 VIM DMA

Fig. 8 illustrates a type-2 VIM DMA apparatus 800 with a type-2 VIM 808 similar to the type-2 VIM apparatus 600 and VIM 616 shown in Fig. 6, but with the addition of DMA apparatus as shown in Fig. 8. In Fig. 8, the VIM is partitioned into a VIM section per functional unit each with duplicate controls and each operated in the same manner. Due to the complexity of Fig. 8, the ALU VIM portion and its associated controls have been broken out and shown in further detail in Fig. 9 for further detailed discussion of its operation. In Fig. 9 the assembly 900 processes the instruction 909 and utilizes a DMA 915 interface and ALU VIM 903 and control inputs from a controls section of the type-2 VIM system. The type-2 VIM DMA apparatus represents an extension to the basic DMA memory interface provided in each PE. The "cycle borrowing" technique is also used in VIM DMA operations. Each preferred VIM portion, for example VIM portion 903, is a two port memory allowing simultaneous read and write accesses.

It is anticipated that in applications where performance is not critical or where simultaneous read and write accesses are deemed not necessary a single shared read/write port VIM can be used.

In the two port design, each port requires its own address and write or read control signals. The write address 931 is sourced from multiplexer 936 since the VIM can be loaded from either instruction path 927 with LV2 instructions or from DMA operations on path 929. The read address 925 is only accessed from the instruction side under control of XV2 instructions. DMA operation consists of a "data" packet made up of SIWs that begin with a LV2 type instruction, followed by the SIWs constituting the specified functional VIM portion to be loaded at the address specified in the LV2 instruction. The type-2 VIM differs from the type-1 VIM in that type-2 functional VIM portions are loaded up to 16 locations at a time while the type-1 VIM loads a single VIM address with the up to 5 SIWs constituting the single VLIW. The type-1 VIM DMA also buffered up a whole VLIW line of SIWs prior to loading the full VLIW in a single cycle to the specified VLIW address. In the type-2 VIM apparatus, it is important to note that the whole block of SIWs for each VIM portion gets loaded prior to that range of VIM addresses which are used by the XV2 instructions. This load priority is necessary because the XV2 instruction allows SIWs to be selected from the different functional VIM portions at different VIM addresses in parallel. The DMA subsystem would be scheduled to transfer a whole block of VIM data for all functional VIM portions. In this manner, the programmer can determine that when the type-2 DMA operation is complete all portions of the VIM are available for program use. The type-2 VIM DMA operation would transfer a single SIW at a time and not buffer up a full VLIW set of multiple SIWs as was done in the type-1 VIM DMA operation.

When this type-1 VIM DMA operation begins, the "data" packet is transferred beginning with the LV2 "data" item over the 32-bit DMA bus 915 and stored into DMA register 1 (DR1) 910. The output 917 of DR1 is provided to VIM controller 922 which provides "cycle borrow" control to the DMA interface. The received LV2 instruction causes the correct Vx base address register to be selected via path 926 and adder 924 adds the base Vx value to the offset found in the received LV2 unit VIM offset field 945 (Fig. 4C bits 7-0 for the ALU defined by bits 17-15). The resultant sum is stored in register 930 for later use. A programmer can use the two Vx base address registers to aid this process where one, say register V0, can be set up for the instruction iVLIW usage and the other, register V1, for DMA base address use. The DMA packet will continue sending the next LV2 SIWs one at a time through DR1 910 to be loaded into the ALU VIM portion. For each SIW received at 910, the SIW is loaded directly into the ALU VIM portion under control of the VIM control 922 at the LV2 instruction specified address for the first SIW and then incrementally as long as the instruction side does not take priority. The DMA VIM control logic automatically increments the VIM address in preparation for the next SIW received on the DMA interface. The VIM controller 922 selects multiplexer 936 for the DMA VIM address 929 and selects the multiplexer 954 to the DR1 register output 917 and in a single cycle the SIW is then loaded into the ALU VIM 903. The DMA interface is then allowed to proceed with the next SIW "data" item of the DMA packet if there is one. It is noted that if the "data" packet is common to multiple PEs, the multiple PEs' functional VIM portions can be loaded in parallel and in synchronism. The DMA operation continues for each functional partition of the full Type-2 VIM apparatus.

Alternate VIM Load Mechanisms via SP or PE Instructions

Two alternate methods for loading VIMs (other than DMA):

(1) SP/PE Store to Special Purpose Register (SSPR) instructions to a specified VIM Port SPR address; and

(2) an SP/PE Load instruction which targets the VPORT MRF register.

Both of these approaches have the following features:

Different LOAD (memory) addresses allow different VLIWs to be loaded into each VIM to support synchronous multiple instruction multiple data (SMIMD) processing setup.

It is not necessary to use instruction memory for LV instructions.

LV instructions can be placed into PE local memories by DMA using both channels for reducing overall VIM load time. DMA can place LV instructions into PE memories in background and VIMs can be loaded in ¼ the time required currently when application needs them.

These approaches also allow the SP or the PEs to reload VIMs while processing other instructions by use of the VLIW architecture.

Another method for loading VIMs in parallel is via PE SSPR instructions. The PE SSPR instruction is basically a variation of the STORE instruction. The SSPR instruction targets a particular address. VLIW instructions (LV followed by a list of instructions) can be loaded into PE memories via DMA, then a LOAD to a PE or SP register followed by an SSPR instruction to a VPORT PE SPR address can be used to load PE VIMs in parallel. This approach has several benefits.

The Load/SSPR data (LV and following instructions) would normally be targeting the instruction register of each PE in the present ManArray architecture, thereby conflicting with subsequent instructions in the pipe. This conflict is solved by providing an alternate decode register (same as with DMA) which would allow an LV instruction with following data to be

decoded in parallel with any other instruction received from the instruction bus. With XVs, this approach requires a dual port VIM. In this approach, loads/stores in an XV can be used to load VIMs with effective throughput of 4 instructions per cycle.

Interrupts must be disabled while VIMs are being loaded. Currently, interrupts are not allowed while processing an LV sequence. In the present architecture, the SP knows what is going on since it decodes the LV instruction. In this approach, there is no signal for PEs writing to a special "VIM load port" using a particular PE SPR address. One solution is to add a signal that is asserted whenever a value is written to the VIM Write Port SPR address of any PE or whenever the VPORT input register (of any PE) has data in it for loading into VIM.

While the present invention has been disclosed in a presently preferred context, it will be recognized that the present teachings may be adapted to a variety of contexts consistent with this disclosure and the claims that follow.